

Key Concepts in Asynchronous Programming in Dart:

Futures:

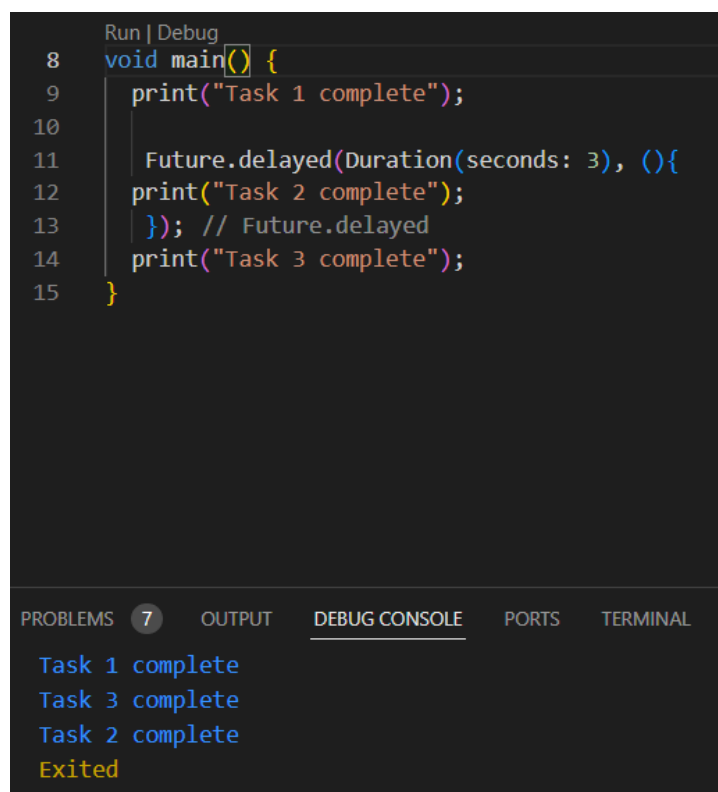
- A Future represents a value that will be available at some point in the future, such as the result of an asynchronous operation.
- A Future can be in two states: uncompleted or completed (with a value or an error)

async/await:

- The `async` keyword is used to mark a function as asynchronous, meaning it can contain `await` expressions.
- The `await` keyword pauses the execution of the function until the Future completes.

step1:

By using future



```
Run | Debug
8 void main() {
9   print("Task 1 complete");
10
11   Future.delayed(Duration(seconds: 3), () {
12     print("Task 2 complete");
13   }); // Future.delayed
14   print("Task 3 complete");
15 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE PORTS TERMINAL

Task 1 complete
Task 3 complete
Task 2 complete
Exited

step 2:

By using `async` and `await`

```

Run | Debug
8 void main() async {
9     print("Task 1 complete");
10
11     await Future.delayed(Duration(seconds: 3), () {
12         print("Task 2 complete");
13     }); // Future.delayed
14
15     //get response from API
16     //print(response);
17     print("Task 3 complete");
18 }
19

```

Streams:

- A Stream provides a sequence of asynchronous data events. Unlike Future, which returns a single value, Stream returns multiple values over time.
- You can listen to a stream using the listen() method or process data with await for in an async function.

```

2
Run | Debug
3 void main() async {
4     await for (var count in countStream(5)) {
5         print(count);
6     }
7 }
8
9 Stream<int> countStream(int max) async* {
10     for (int i = 0; i <= max; i++) {
11         await Future.delayed(Duration(seconds: 1));
12         yield i;
13     }
14 }
15

```

PROBLEMS 7 OUTPUT DEBUG CONSOLE PORTS TERMINAL

```

0
1
2
3
4
5
Exited

```

Mixins:

Mixins in Dart allow you to reuse code across multiple classes. They are a way to implement shared functionality without using inheritance. To define a mixin, use the `mixin` keyword. Classes can use one or more mixins using the `with` keyword.

```
1  mixin Swimmable {
2      void swim() {
3          print("Swimming...");
4      }
5  }
6
7  mixin Flyable {
8      void fly() {
9          print("Flying...");
10     }
11 }
12
13 class Duck with Swimmable, Flyable {}
14
15 class Fish with Swimmable {}
16
17 Run | Debug
18 void main() {
19     var duck = Duck();
20     duck.swim(); // Output: Swimming...
21     duck.fly();  // Output: Flying...
22
23     var fish = Fish();
24     fish.swim(); // Output: Swimming...
25 }
```